DL-HW #11

2015004693_양상헌

실행환경: MAC OS TERMIINAL(MACBOOK PRO 2015 RETINA , MOJAVE 10.14.6) , ANACONDA

1. Source Code:

- *Assignment11 폴더 참조*
  (DL_HW_11week.py)

```
#--------------- attention --------------#
# alpha dimmension : matmul ((B,S1,H), (B,H,S2)) = (B,S1,S2)
alphas =  tf.nn.softmax(  tf.matmul(enc_outputs, tf.transpose( dec_outputs ,  perm = [0,2,1]))  )
#print(alphas)
# attention dimmention: (B, S2 S1) * (B,S1,H) = (B, S2, H)
attention = tf.matmul(tf.transpose(alphas,perm = [0,2,1] ), enc_outputs)
#print(attention)
# (B, S2, H)concat( B, S2, H) = (B, S2, 2*H)
outputs = tf.concat([dec_outputs, attention], 2)
#print(outputs)

model = tf.layers.dense(outputs, n_class, activation=None)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=model, labels=targets))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

#----------------------------------------------#
```

11주차 실습 자료에 나온 SEQ2SEQ 모델에 간략하게 Attention을 추가해 주었다.

2. Result:

```
2020-06-03 23:17:57.690572: I tensq
thread pool with default inter op s
 performance.
Epoch: 0001 cost = 3.787066
Epoch: 0002 cost = 2.879569
Epoch: 0003 cost = 1.806535
Epoch: 0004 cost = 1.385661
Epoch: 0005 cost = 0.987185
Epoch: 0006 cost = 0.555801
Epoch: 0007 cost = 0.326304
Epoch: 0008 cost = 0.225372
Epoch: 0009 cost = 0.148138
Epoch: 0010 cost = 0.241220
Epoch: 0011 cost = 0.195021
Epoch: 0012 cost = 0.213796
Epoch: 0013 cost = 0.141581
Epoch: 0014 cost = 0.179482
Epoch: 0015 cost = 0.044785
Epoch: 0016 cost = 0.088500
Epoch: 0017 cost = 0.013932
Epoch: 0018 cost = 0.070279
Epoch: 0019 cost = 0.020840
Epoch: 0020 cost = 0.030914
Epoch: 0021 cost = 0.045978
Epoch: 0022 cost = 0.023420
Epoch: 0023 cost = 0.038643
Epoch: 0024 cost = 0.044644
Epoch: 0025 cost = 0.025336
Epoch: 0026 cost = 0.004727
Epoch: 0027 cost = 0.081906
Epoch: 0028 cost = 0.004071
Epoch: 0029 cost = 0.001210
Epoch: 0030 cost = 0.008896
Epoch: 0031 cost = 0.026600
Epoch: 0032 cost = 0.014886
Epoch: 0033 cost = 0.002385
Epoch: 0034 cost = 0.003728
Epoch: 0035 cost = 0.001666
```

```
Epoch: 0073 cost = 0.000871
Epoch: 0074 cost = 0.001839
Epoch: 0075 cost = 0.000206
Epoch: 0076 cost = 0.000894
Epoch: 0077 cost = 0.000380
Epoch: 0078 cost = 0.000108
Epoch: 0079 cost = 0.000848
Epoch: 0080 cost = 0.000243
Epoch: 0081 cost = 0.000098
Epoch: 0082 cost = 0.000268
Epoch: 0083 cost = 0.000500
Epoch: 0084 cost = 0.000560
Epoch: 0085 cost = 0.000590
Epoch: 0086 cost = 0.000335
Epoch: 0087 cost = 0.000320
Epoch: 0088 cost = 0.001310
Epoch: 0089 cost = 0.000386
Epoch: 0090 cost = 0.000727
Epoch: 0091 cost = 0.000869
Epoch: 0092 cost = 0.000152
Epoch: 0093 cost = 0.000279
Epoch: 0094 cost = 0.000313
Epoch: 0095 cost = 0.001301
Epoch: 0096 cost = 0.000057
Epoch: 0097 cost = 0.000345
Epoch: 0098 cost = 0.001404
Epoch: 0099 cost = 0.000253
Epoch: 0100 cost = 0.000508
OPTIMIZATION IS DONE!!


=====Translation TEST=====
word ->  단어
wodr ->  나무
love ->  사랑
loev ->  사랑
abcd ->  놀이
(tensorflow) yangsangheonui-MacBook
```

3. Disscussion:

< 코드 설명 >

```
#--------------- attention --------------#
# alpha dimmension : matmul ((B,S1,H), (B,H,S2)) = (B,S1,S2)
alphas =  tf.nn.softmax(  tf.matmul(enc_outputs, tf.transpose( dec_outputs ,  perm = [0,2,1]))  )
#print(alphas)
# attention dimmention: (B, S2 S1) * (B,S1,H) = (B, S2, H)
attention = tf.matmul(tf.transpose(alphas,perm = [0,2,1] ), enc_outputs)
#print(attention)
# (B, S2, H)concat( B, S2, H) = (B, S2, 2*H)
outputs = tf.concat([dec_outputs, attention], 2)
#print(outputs)

model = tf.layers.dense(outputs, n_class, activation=None)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=model, labels=targets))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

#----------------------------------------------#
```

소스코드의 전체적인 구조는 11주차 실습 자료에 나온 seq2seq 모델의 구조를 변경하지 않고 Attention기법만 추가한 구조이고, 실제로 매우 짧은 코드만 추가하여 구현하였다. 이 과제에서 사용된 Attention 매커니즘은 가장 단순한 방법 중 하나인 백터 내적을 이용 하는 것이다.

모든 인코더의 은닉 상태들의 정보를 가져와서 현재 디코더의 은닉 상태와 백터 내적을 진행하고, 이를 Softmax 활성화 함수를 사용하여 알파 값으로 만들어주었다. 이후에 각각의 인코더의 은닉 상태들 마다 생성된 알파 값을 곱한 다음 모두 더해주었다. 이렇게 만들어진 벡터가 바로 Attention 벡터이고 이를 디코더의 아웃풋에 tf.concat을 이용하여 붙여주었다. 기존의 디코더 아웃풋에는 Attention에 관련된 정보를 추가해 주었고, 이 Attention정보가 추가된 디코더의 아웃풋을 이용하여 모델을 동작하도록 하였다.

사용된 Attention 메커니즘 자체의 수학식은 복잡해 보일 수 있지만 위의 3줄의 Matrix 연산으로 이를 표현하는 것이 가능하다. 백터 내적을 행렬 곱으로 표현하는데 매우 큰 어려움이 있었고, Matrix 곱셈에서 차원을 맞춰주는 부분도 많은 시간이 들었다. (추가된 코드는 매우 짧지만, 지금까지 했던 과제들 중 가장 어려웠고, 가장 오랜시간이 걸린 과제이다.)

< 결과 분석 >

많은 횟수를 실행해 보았을 때, 평균적으로 기존의 seq2seq보다 더 높은 성능을 내는 것을 확인할 수 있었다. 기존의 Seq2seq 모델은 거리가 멀어질 수 록 결과값에 영향을 미치는 부분이 약해지지만, Attention 값을 추가해줌으로써 거리에 상관없이 더 중요한 단어를 선택하여 결과값을 만들어 주게 됨으로 더 좋은 결과가 나오는 것을 확인할 수 있다.